Yining Hua
John Foley
Data Structure
November 15, 2018

CSC212P8 SpellChecker

REFLECTION

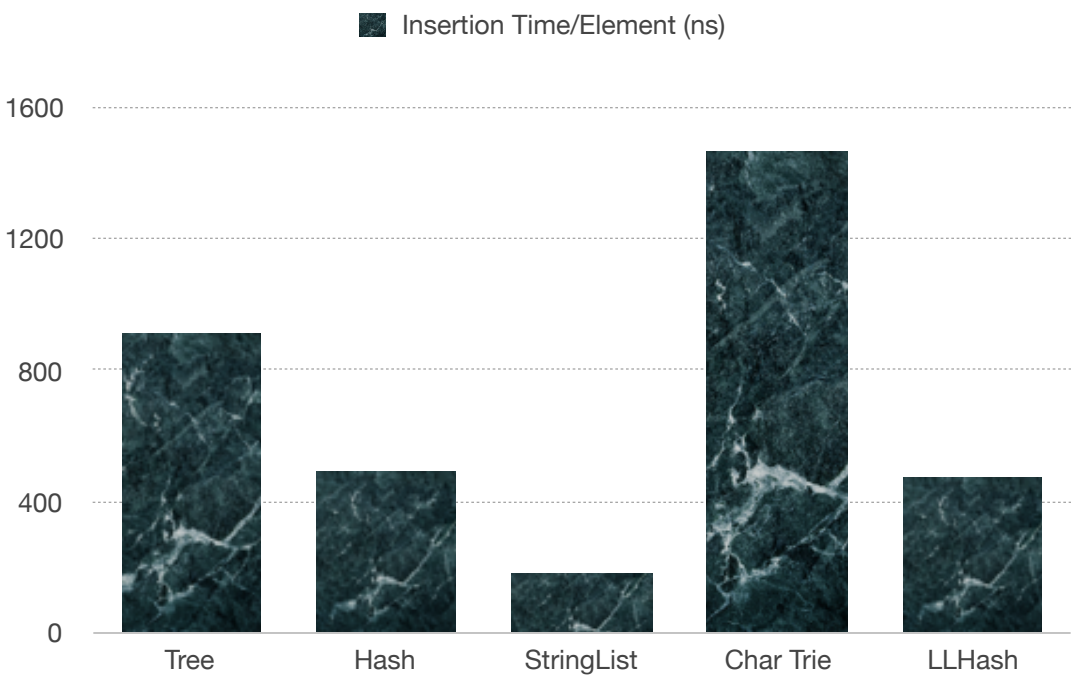
1. Learned how hash set, tree set, trie and hash set with manipulatable buckets work.
    1. All elements in hash set has its hash value.
    2. Hash set might collide.
    3. Increasing the number of buckets can help with that.
    4. Which is a tradeoff between space and time.
    5. Tree set can behave like array list if the root is a bad one and so the tree does not have branches..
    6. Trie can be used to store vocabularies by storing each character and indicating what the following character is, which nodes are terminal nodes.

2. Learned how to test running times for functions
    1. System.nanoTime()

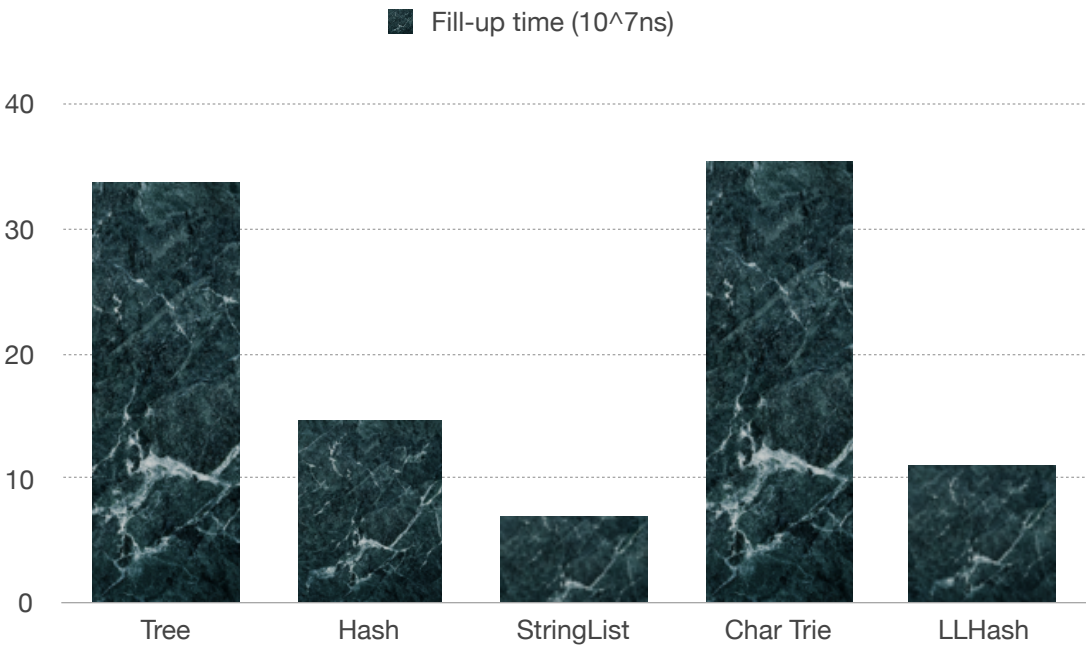
3. Leaned why binary search was wrong for so long
    1. Consider about overflow.
    2. Bitwise operation or low+(high-low)/2 can help.

4. Java Regex
    1. private static Pattern spacesOrPunctuation = Pattern.compile("(\\s+|\\p{Punct})");
    2. To get rid of integers: \\d+

1.How long does it take to fill each data structure?



2. Plot insertion time per element for each of these data structures.



3. Is there a timing difference between constructing `HashSet` and `TreeSet` with their input data or calling add in a for loop? If yes, use the words "balancing" and "resizing" to explain what's going on.

Building time - tree: 182368704
Building time - tree by loop: 128705347
Building time - hash: 50928600
Building time - hash by loop: 156088831

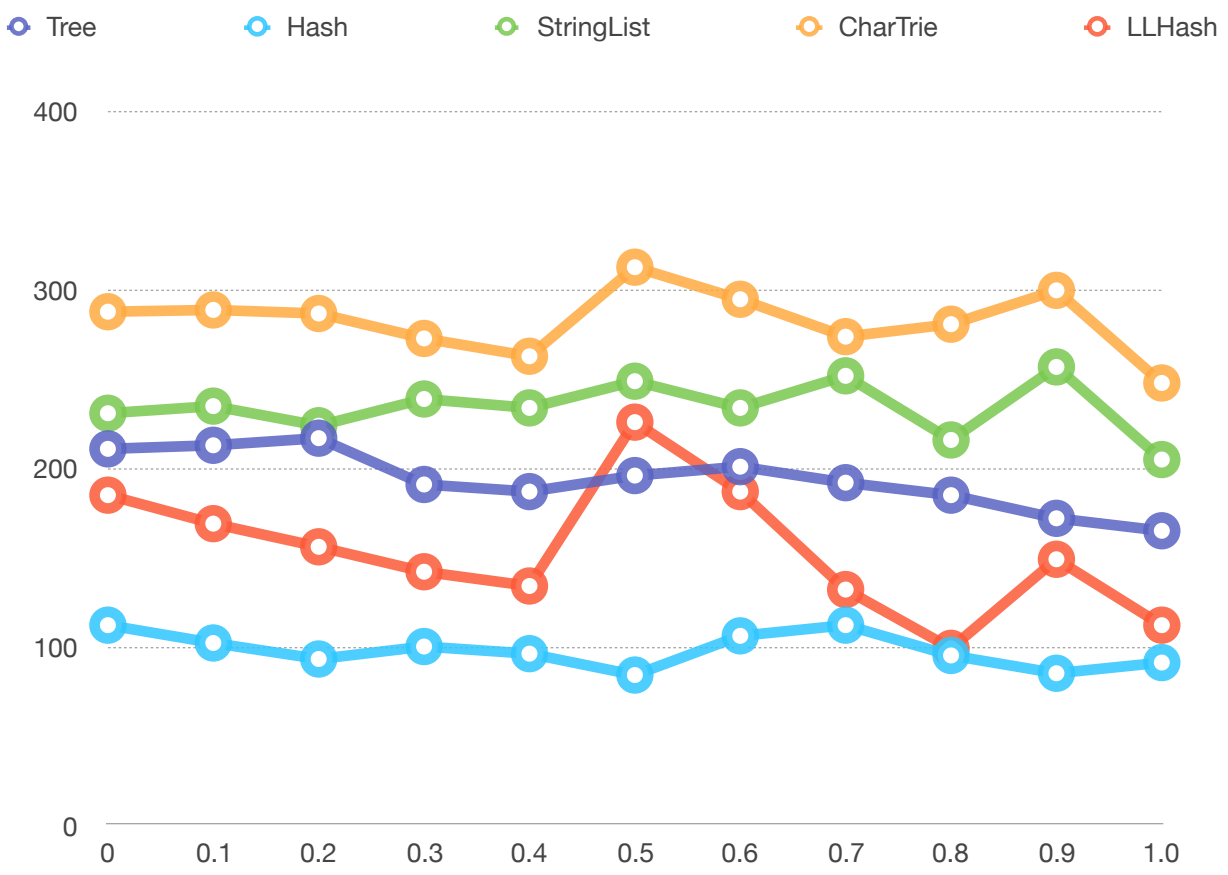
The time needed for constructing a tree is longer than that for constructing a hash because in order to add a new element in a tree we have to compare the value of the new element to elements which are already in the tree and so to get the position the new element should be inserted. But when adding a new element to hash, we simply calculate the new element's hash value and put it into its corresponding bucket.

The time needed for constructing a tree with input data is shorter than that by calling add in a for loop. Theoretically speaking I think it should have been the opposite. Since when given the whole input data Java will "sort the data and insert in a pre-balanced order into the TreeSet". Then when given the input one element by one element, Java can no longer pick such guess.

The time needed for constructing a Hash Set with input data is longer than calling add. Because Java will "pick a good guess for the number of buckets needed for the words given to it". If we call add on each element Java has to resize the bucket with more elements added, thus resulting a longer construction time.

(quoted lines from JJ)

4. Construct a dataset that has Strings that are both in and not in the dictionary.



5. What is the ratio that's “mis-spelled"?

Not found: 9892

Book size: 48363

Misspelled ratio: 0.20453652585654322

6. Are the query speeds the same over real-world data?

Real-world data takes longer.

7. What are some of the words that are “mis-spelled"?

Misspelled words are generally words with s/ed/es at the end, which are variants.

Also foreign words and words being segmented in between are some of the “mis-spelled” words.

8. Double-check that your query speeds have not changed with your implementation of binary search. If they have, why might that be?

Collections.binarySearch()->the time to look up one item is around 700-1000ns.

binarySearch() -> similarly the time is around stays around 700-1000ns

But generally my version of binarySearch takes longer to find an item. Maybe the Collection.binarySearch() function is better implemented than mine.

9. Play with the size of LLHash. Does this change your perception of its speed?

The larger the LLHash size, the longer the time needed for building the data structure and the time needed for inserting one element, and the shorter the time needed for looking up an element since there are more buckets and each buckets contains fewer elements.

Conclusion:

From the graph above we can conclude that Hash is the best data structure for spell-checking. This is because of the way Hash is implemented: each element has a hash value and can be retrieved by its hash value. LLHash does a slightly worse job than Hash when compared with other data structures, so it can also be a nice choice for spellchecking.